Aprendendo R em 10 minutos

Vanderlei Júlio Debastiani (vanderleidebastiani@yahoo.com.br)

05 Janeiro 2019

Índice de conteúdo

1 Introdução

R é um ambiente de programação voltado para manipulação, análise de dados e apresentação gráfica. R é um projeto em constante desenvolvimento resultado de esforços colaborativos com contribuições de todo o mundo. Disponibiliza uma ampla variedade de métodos estatísticos e gráficos, sendo ainda facilmente extensível através de pacotes ou funções desenvolvidas pelos próprios usuários. O R possue uma estrutura de código aberto (*open source*) com linguagem de programação multiparadigma, sendo que tarefas computacionais intensivas podem ser realizadas por meio de integração de códigos em C, C++, Java, Python e Fortran. O

R tem a sua própria documentação em formato semelhante a LaTeX, que é usada como documentação das funções.

Este texto trata dos principais aspectos da linguagem R e tentará ensinar R em 10 minutos. Obviamente que não é possível aprender uma linguagem de programação em apenas alguns minutos, para realmente aprender R é preciso programar e exercitar por um tempo. O texto mostrará alguns conceitos básicos para começar, como sintaxe, operações, tipos de dados, estruturas, declarações de controle de fluxo e funções. A ideia é que o leitor seja capaz de reproduzir os comandos aqui contidos e exercitar os conceitos apresentados. Algumas conceitos serão introduzidas diretamente no código e apenas brevemente comentados. Ao final também é apresentado um guia de ajuda rápida mostrando as principais palavras da sintaxe, operadores e funções da linguagem R apresentados aqui.

2 Propriedades

R possui fortes recursos de programação orientada por objetos, sendo que tudo é considerado objeto. A tipagem é dinâmica, não exigingo declarações prévias do tipos de dados do objeto. É uma linguágem sensível à caixa das letras (*case-sensitive*), ou seja, *var* e *VAR* são consideradas duas variáveis diferentes.

3 Obtendo ajuda

A ajuda está sempre disponível diretamente no interface. O funcionamento da função pode ser obtido usando as funções **help** (atalho **?**). Argumentos das funções podem ser obtidos com a função **args**, classes dos objetos pela função **class** e estrutura dos objetos pela função **str**. Ainda é possivel pesquisar por assunto, usando a função **help.search** (atalho **??**).

```
help(abs)
class(5)
[1] "numeric"
str(5)
 num 5
```

4 Sintaxe

O R não possui caracteres de terminação de instrução obrigatórios e os blocos são especificados por chaves (**{}**). Os comentários começam com sustenido (#) e não há sintaxe para comentários em multiplas linhas. Valores são atribuídos (na verdade, objetos são vinculados a nomes) com os sinais de **"<-"**, **"="** ou **"->"** (mais usado é o **"<-"** enquanto que **"="** é obrigátorio quando usado para especificar argumentos das funções). Os indentificadores das variáveis (nomes dos objetos) podem ser uma combinação de letras, dígitos, ponto (.) ou underline (_), devendo começar com letras ou digítos. Quando começam com um ponto, não poderão ser seguido por um dígito. O teste de igualdade é feito usando dois sinais de igual (**"=="**). Não há operadores de incremendo direto. As vírgulas (**,**) separam argumentos em uma função, sendo que é possível usar várias variáveis em uma linha. Outras informações sobre sintaxe são encontradas em *?Syntax*.

```
minha.var <- 2 # Separador "." é preferível ao identificar os objetos
minha.var
[1] 2
meu.string <- "Hello"
meu.string
[1] "Hello"
print(meu.string)
[1] "Hello"
# Isso é um comentário
minha.var <- meu.string
```

```
minha.var
[1] "Hello"
```

5 Tipos de dados

Os tipos de dados podem ser numéricos (*numeric*), caracter (*character*) ou lógicos (*logical*). Os dados do tipo *numeric* são número inteiros, reais e complexos. Podem serem escritos também com notação científica, utilizando o **E** (ou **e**), se numéricas são seguidos pela letra **L** são considerados inteiro e se seguidos pela letra **i** são consideradas complexos. O tipo *character* armazena caracteres ou texto e são escritos obrigatoriamente entre aspas duplas (**""**) ou simples (**''**). As dados boleanos (verdadeiros ou falsos) são definidos como **TRUE** e **FALSE** (abreviação **T** e **F** respectivamente). A função **typeof** mostra o tipo de modo de armazenamento de qualquer objeto. A conversão entre tipos é livre feita pelas funções de prefixo **as.** (por exemplo *as.numeric()*) e a checagem lógica feita pelas funções de prefixo **is.** (por exemplo *is.numeric()*).

```
var.numerica.1 <- 5.2
var.numerica.1
[1] 5.2
class(var.numerica.1)
[1] "numeric"
var.numerica.2 <- 5E6 # mesmo que 5e6
var.numerica.2
[1] 5e+06
typeof(var.numerica.2)
[1] "double"
class(var.numerica.2)
[1] "numeric"
var.caracter <- "var"
var.caracter
[1] "var"
class(var.caracter)
[1] "character"
var.logica <- TRUE
var.logica
[1] TRUE
class(var.logica)
[1] "logical"
as.character(var.numerica.1)
[1] "5.2"
as.numeric(var.caracter)
Warning: NAs introduced by coercion
[1] NA
is.numeric(var.caracter)
[1] FALSE
```

6 Operações aritméticas e lógicas

As operações aritméticas básicas e testes lógicos são disponíveis. Mais informações sobre operações são encontradas em *?Arithmetic*, *?Comparison* e *?Logic* .

```
2+2 # Soma
[1] 4
8-3 # Subtração
```

```
[1] 5
3*8 # Multiplicação
[1] 24
8/2 # Divisão
[1] 4
2^8 # Potências
[1] 256
(2+4)/7 # Prioridades de solução. Diferente de 2+4/7
[1] 0.8571429
10%/%3 # Parte inteira da divisão. O inteiro da divisão de um número por outro
[1] 3
10%%3 # Módulo. O resto da divisão de um número por outro
[1] 1
matrix(c(1, 2, 3),
       nrow = 3,
       ncol = 1)%*%matrix(c(4,2,3),
                          nrow = 1,
                          ncol = 3) # Produto de matrizes
     [,1] [,2] [,3]
[1,]    4    2    3
[2,]    8    4    6
[3,]   12    6    9
matrix(c(1, 2, 3),
       nrow = 1,
       ncol = 3)%o%matrix(c(4,2,3),
                          nrow = 1,
                          ncol = 3) # Produto diádico
, , 1, 1

     [,1] [,2] [,3]
[1,]    4    8   12

, , 1, 2

     [,1] [,2] [,3]
[1,]    2    4    6

, , 1, 3

     [,1] [,2] [,3]
[1,]    3    6    9
matrix(c(1, 3, 2, 4),
       nrow = 2,
       ncol = 2)%x%matrix(c(0, 6, 5, 7),
                          nrow = 2,
                          ncol = 2) # Produto de Kronecker
     [,1] [,2] [,3] [,4]
[1,]    0    5    0   10
[2,]    6    7   12   14
[3,]    0   15    0   20
[4,]   18   21   24   28
c("a", "b", "c")%in%c("b", "c") # Operador de correspondência
[1] FALSE  TRUE  TRUE
```

```
2<3 # Comparar, menor
[1] TRUE
3>3 # Comparar, maior
[1] FALSE
3<=3 # Comparar, menor ou igual
[1] TRUE
2>=3 # Comparar, maior ou igual
[1] FALSE
3==3 # Comparar, exatamente igual
[1] TRUE
2!=3 # Comparar, diferente
[1] TRUE
!3==3 # Lógico, NÃO. Inverter resultado de teste lógico
[1] FALSE
3==3 & 3!=3 # Lógico, critério aditivo E. Operação elementar
[1] FALSE
3==3 | 3!=3 # Lógico, critério aditivo OU. Operação elementar
[1] TRUE
3==3 && 3!=3 # Lógico, E.
[1] FALSE
3==3 || 3!=3 # Lógico, OU.
[1] TRUE
```

Operadores **&** e | fazem a operação elementar produzindo resultados de comprimento do operador mais longo. Já **&&** e || examinam apenas o primeiro elemento dos operadores e como resultado apenas um único valor lógico é retornado. Zero é sempre considerado *FALSE* e números diferentes de zero são considerados *TRUE*.

```
x <- c(TRUE, FALSE, 0, 6)
x
[1] 1 0 0 6
y <- c(FALSE, TRUE, FALSE, TRUE)
y
[1] FALSE  TRUE FALSE  TRUE
!x
[1] FALSE  TRUE  TRUE FALSE
x&y
[1] FALSE FALSE FALSE  TRUE
x&&y
[1] FALSE
x|y
[1]  TRUE  TRUE FALSE  TRUE
x||y
[1] TRUE
```

7 Tipos de estruturas

As estruturas básicas de dados disponíveis são vetores (*vector*), fatores (*factor*), data.frames (*data.frame*), matrizes (*matrix*) e listas (*list*). Os vetores e fatores são conjuntos de dados unidimensionais do mesmo tipo. Os vetores podem conter dados lógicos (*logical*), inteiros (*integer*), reais (*double*), caracteres (*character*) ou complexos (*complex*), são geralmente concatenados pela função (**c**) e sempre separados por vírgulas (**,**). Os fatores são estruturas de dados usada para dados (*numeric* ou *character*) com um número finito (predefinido) de categorias, podendo estas serem ordenadas ou não. Fatores são definidos pelas função **factor** e **ordered**.

Os data.frames são conjuntos bidimensionais de vetores de mesmo comprimento sendo que cada vetor pode ser de tipos diferentes. Os data.frames são criados pela função (**data.frame**) onde os vetores são agrupados nas coluna e o conteúdo dos vetores formam as linhas. Por padrão, a função converte os vetores de caracteres (*character*) em fatores não ordenados (*factor*). As matrizes também são conjuntos bidimensionais de dados do mesmo tipo, podem ser criadas com a função **matrix** ou com a combinação de vetores com as função **cbind** e **rbind** (concatena os vetores por coluna ou por linhas respectivamente). A listas são conjuntos de dados de qualquer tipo (incluindo listas de listas), são criadas com a função **list**. Vetores, fatores e listas podem conter nomes (acessados ou atribuidos pela função **names**) e possuem como atributo o comprimento (obetidos pela função **length**). Os data.frames e matrizes podem conter nomes para linhas e colunas (acessados ou atribuidos pelas funções **rownames** e **colnames**, respectivamente) e contêm como atributos os números de colunas e linhas (obetidos pelas funções **ncol** e **nrow**, respectivamente). O índice do primeiro item é sempre o zero (0). Os intervalos das estruturas podem ser acessadas usando dois pontos (**:**). Os índice com números negativos retornam todos menos os especificados. A indexação também ser realizada com vetores lógicos e com nomes nos casos de estruturas nomeadas. A função **str** exibir resumidamente a estrutura interna dos objeto no R. A conversão entre estruturas é livre feita pelas funções de prefixo **as.** (por exemplo *as.matrix()*) e a checagem lógica feita pelas funções de prefixo **is.** (por exemplo *is.data.frame()*).

```
# Estruturas
vetor <- c("a", "b", "c", "d")
vetor
[1] "a" "b" "c" "d"
class(vetor)
[1] "character"
names(vetor)
NULL
names(vetor) <- c("letra1", "letra2", "letra3", "letra4")
vetor
letra1 letra2 letra3 letra4
   "a"    "b"    "c"    "d"
fator <- factor(c("a", "b", "c"))
fator
[1] a b c
Levels: a b c
levels(fator)
[1] "a" "b" "c"
fator.ordenado <- ordered(c("a", "b", "c"))
fator.ordenado
[1] a b c
Levels: a < b < c
dataframe <- data.frame(vetor.de.caracteres = vetor, vetor.numerico = 1:4)
dataframe
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4
str(dataframe) # Por padrão a função converte vetores de caracteres em fatores
'data.frame':   4 obs. of  2 variables:
 $ vetor.de.caracteres: Factor w/ 4 levels "a","b","c","d": 1 2 3 4
 $ vetor.numerico     : int  1 2 3 4
ncol(dataframe)
[1] 2
nrow(dataframe)
[1] 4
```

```
matriz <- matrix(data = 1:20, nrow = 5, ncol = 4) # cbind e rbind também podem ser usadas
matriz # Por padrão os dados são organizados por colunas
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
str(matriz) # Apenas uma tipo de dado é permitido
 int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
lista <- list(1, dataframe, matriz, list("outra", "lista"))
lista
[[1]]
[1] 1

[[2]]
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4

[[3]]
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20

[[4]]
[[4]][[1]]
[1] "outra"

[[4]][[2]]
[1] "lista"
names(lista)
NULL
names(lista) <- c("l1", "l2", "l3", "l4")
lista
$l1
[1] 1

$l2
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4

$l3
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
```

```
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20

$14
$14[[1]]
[1] "outra"

$14[[2]]
[1] "lista"
length(lista)
[1] 4

# Indexação das estruturas
vetor[1] # Primeiro elemento
letra1
   "a"
vetor[c(FALSE, FALSE, TRUE, FALSE)] # Indexação com valores lógicos
letra3
   "c"
vetor[1:3] # Primeiro a terceiro elementos
letra1 letra2 letra3
   "a"    "b"    "c"
vetor[] # Todos
letra1 letra2 letra3 letra4
   "a"    "b"    "c"    "d"
vetor[-1] # Todos, exceto o primeito elemento
letra2 letra3 letra4
   "b"    "c"    "d"
vetor[c(1, 4)] # vetor[1, 4] não funciona
letra1 letra4
   "a"    "d"
vetor["letra4"] # Pelo nome
letra4
   "d"
vetor[1] <- "z" # muda o valor do item 1
vetor
letra1 letra2 letra3 letra4
   "z"    "b"    "c"    "d"
vetor <- NULL # Vetor deletado
vetor
NULL

fator
[1] a b c
Levels: a b c
fator[1] <- "c" # mudar de fator
fator # níveis permanecem
[1] c b c
Levels: a b c
fator[1] <- "z" # fator inválido
Warning in `[<-.factor`(`*tmp*`, 1, value = "z"): invalid factor level, NA
```

```
generated
fator # NA é gerado
[1] <NA> b    c
Levels: a b c

dataframe[1] # Mostra primeiro vetor (coluna)
       vetor.de.caracteres
letra1                   a
letra2                   b
letra3                   c
letra4                   d
dataframe$vetor.de.caracteres # Indexação pelo nome
[1] a b c d
Levels: a b c d
dataframe[1, ] # Linha
       vetor.de.caracteres vetor.numerico
letra1                   a              1
dataframe[, 2] # Coluna. Por padrão a estrutura é convenvertida em vetor...
[1] 1 2 3 4
dataframe[, 2, drop = FALSE] # ... Mantém a estrutura de data.frame
       vetor.numerico
letra1              1
letra2              2
letra3              3
letra4              4
dataframe[,] # Tudo
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4
dataframe[, 2] <- NULL # Apenas colunas podem ser deletadas
dataframe$novo.vetor <- c("a", "b", "c", "d") # Adicionar vetor...
str(dataframe) # ... nesse caso a classe permanece
'data.frame':   4 obs. of  2 variables:
 $ vetor.de.caracteres: Factor w/ 4 levels "a","b","c","d": 1 2 3 4
 $ novo.vetor         : chr  "a" "b" "c" "d"

matriz[1, ]
[1]  1  6 11 16
matriz[, 2]
[1]  6  7  8  9 10
matriz[, 2, drop = FALSE] # Matriz 1x5, não um vetor
     [,1]
[1,]    6
[2,]    7
[3,]    8
[4,]    9
[5,]   10
matriz[1, 4] # Um elemento da matriz
[1] 16
matriz[1, 4] <- 0 # Nada pode ser deletado, apenas alterado de valor (incluindo NA)
```

```
str(lista)
List of 4
 $ l1: num 1
 $ l2:'data.frame': 4 obs. of  2 variables:
  ..$ vetor.de.caracteres: Factor w/ 4 levels "a","b","c","d": 1 2 3 4
  ..$ vetor.numerico     : int [1:4] 1 2 3 4
 $ l3: int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
 $ l4:List of 2
  ..$ : chr "outra"
  ..$ : chr "lista"
lista[2] # Acessar sublista
$l2
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4
lista[[2]] # Acessar conteúdo da lista
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4
lista[1:3] # Acessar várias sublistas. lista[[1:3]] não funciona
$l1
[1] 1

$l2
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
letra3                   c              3
letra4                   d              4

$l3
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
lista[["l3"]][1,3] # Usando nomes e elemento da matriz
[1] 11
lista[["l3"]] <- NULL # Deletar item da lista. Mesmo que lista["l3"], lista[3], lista[[3]]
lista$novoitem <- 1:3 # Adicionar item a lista
lista
$l1
[1] 1

$l2
       vetor.de.caracteres vetor.numerico
letra1                   a              1
letra2                   b              2
```

```
letra3                  c             3
letra4                  d             4

$l4
$l4[[1]]
[1] "outra"

$l4[[2]]
[1] "lista"


$novoitem
[1] 1 2 3
```

8 Declarações de controle de fluxo

As declarações de controle de fluxo são realizadas por **if**, **else**, **while**, **repeat**, **for**, **break** e **next**. As funções **ifelse** e **switch** também podem ser utilizadas para controle de fluxo. Sequências de inteiros podem ser obtidas pelas funções, **seq**, **rep** ou **seq_len** ou usando **:** (por exemplo *seq_len(8)* ou *1:8* para uma sequencia de 1 a 8). Outras informações sobre controle de fluxo são encontradas em *?Control*.

```
minha.var <- -10
minha.var
[1] -10

if(minha.var == -10) # Condicional SE
  # Se a condição só tiver uma linha não precisa das chaves {}
  # Linhas de comentários não entram na contagem
  print("minha.var é -10")
[1] "minha.var é -10"


if(minha.var>=0){ # Condicional SE...
  print("minha.var é positiva ou zero")
} else { # ... SENÃO (caso contário)
  print("minha.var é negativa")
} # Melhor usar chaves {} em todas as afirmações
[1] "minha.var é negativa"


minha.var
[1] -10
while(minha.var<50){ # Repete ENQUANTO afirmação não for TRUE (verdadeira)
  minha.var <- minha.var+1
}
minha.var
[1] 50


repeat { # REPETIR até ...
  minha.var <- minha.var+1
  if(minha.var>100) {
    break # PARAR
```

```
  }
}
minha.var
[1] 101


for(i in 1:5){ # LOOP (LAÇOS). A sequêancia pode ser outra variável
  print(i)
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5

for(i in 1:5){
  if(i == 3)
    next # PASSA para a próxima sequência do loop
  print(i)
}
[1] 1
[1] 2
[1] 4
[1] 5

for(i in 1:5){
  if(i == 3)
    break # PARAR loop
  print(i)
}
[1] 1
[1] 2
```

9 Funções

As funções são declaradas com a palavra **function**. Outros argumentos, obrigatórios ou opcionais, são definidos na declaração da função entre parênteses (()), podendo estes conter valores predefinidos (*default*). Os argumentos das funções seguem a ordem especificada na declaração da função, podendo ou não ser nomeados ao atributi o valor. Se argumentos são nomeados não precisam seguir a ordem predefinida. A função **return** especifica o que é retornado. A função só pode retornar um objeto, mas pode ser uma lista com vários valores. Uma função pode ser recursiva (*recursive*), chama a si mesma, podendo ser usada para resolver problemas dividindo-os em subproblemas menores e mais simples. É possível definir operações com notação infixa (*infix operator*). Para definição de operações infixas a definição das funções devem iniciar e terminar % e devem ser usadas aspas simples (") para indicar notação de símbolo especial. Mais informações sobre funções são encontradas em *¿"function".*

```
minha.funcao <- function(x){
  x <- x+1
  return(x)
}
minha.funcao(3) # mesmo que minha.funcao(x = 3)
[1] 4
```

```r
# Os argumentos somar e subtrair são opcionais.
# Ambos possuem 0 como valor predefinido (default)
minha.segunda.funcao <- function(x, somar = 0, subtrair = 0){
  x.somado <- x+somar
  x.subtraido <- x-subtrair
  res <- list(x = x, somado = x.somado, subtraido = x.subtraido)
  return(res) # Retorna apenas um valor, no exemplo a lista
}
minha.segunda.funcao(1)
$x
[1] 1

$somado
[1] 1

$subtraido
[1] 1
minha.segunda.funcao(x = 1,
                     soma = 2,
                     subtrair = 0) # mesmo que minha.segunda.funcao(1, 2, 0)
$x
[1] 1

$somado
[1] 3

$subtraido
[1] 1

# Quando especificados os argumentos não precisam seguir a ordem predefinida
minha.segunda.funcao(1,
                     subtrair = 3)
$x
[1] 1

$somado
[1] 1

$subtraido
[1] -2

`%divisible%` <- function(x,y)
{
  if (x%%y ==0) return (TRUE)
  else          return (FALSE)
}

6%divisible%3
[1] TRUE
```

10 Ambiente e escopo das variáveis

O R usa o conceito de ambiente (*environment*) para armazenar as variáveis (objetos e funções por exemplo). O principal ambiente (global) é denominado *R_GlobalEnv* (*.GlobalEnv* nos códigos) e definido automaticamente ao abrir o programa. Podem existir múltiplos ambientes simultaneamente, sendo que na definição de novas funções um novo ambiente é criado para armazenar as variáveis locais. Existe uma distinção entre variáveis globais e locais, que são armazenadas em diferentes ambientes. Variáveis globais são aquelas variáveis que existem durante a execução de um programa, podem ser alteradas e acessadas por qualquer parte do programa. Variáveis locais são aquelas que existem apenas dentro de uma certa parte de um programa, são criadas e destruídas quando a função termina. No entanto, variáveis globais e locais dependem da perspectiva de uma função, já que os ambientes podem estar alinhadas dentro de outros ambientes. A função **ls** lista os objetos do ambiente atual e a função **rm** remove os objetos. As variáveis globais podem ser lidas mas não podem ser atribuídas com a atribuição padrão (<-). A atribuição padrão serve apenas para atribuir variáveis locais. Para fazer atribuições a variáveis globais, o operador de super atribuição (<<-) é usado. Ao usar este operador dentro de uma função, ele procura pela variável no ambiente do nível superior, se não for encontrado, continua procurando o próximo nível até atingir o ambiente global. Se a variável ainda não for encontrada, ela será criada e atribuída no nível global.

```
var1 <- 1
var2 <- 2
funcao <- function(x) {
    x <- 0
    return(x)
}
ls() # O objeto x (arguemnto da função) não esta no ambiente global, é uma variável local
[1] "funcao" "var1"   "var2"
environment()
<environment: R_GlobalEnv>

var1 <- 1
funcao <- function() {
    var1 <- 0 # Uma nova variável local var1 é criada
    print(var1)
}
funcao() # Mostra valor da variável local (var1) dentro da função
[1] 0
var1 # Valor da variável var1 fora da função (global) não é alterado
[1] 1

funcao <- function() {
    var1 <<- 0 # Atribuir o valor zero para a variável global
    var3 <<- 0 # Se variavel global não exisi, uma é criada
    print(var1)
}
funcao() # Mostra valor da variável local (var1) dentro da função
[1] 0
var1 # Note que o valor da variável var1 fora da função (global) foi alterado
[1] 0
var3 # Variável var3 não existiam, mas foi criada pela função
[1] 0

funcao.exterior <- function(x.exterior){
    funcao.interior <- function(x.interior){
        print("Dentro da funcao.interior:")
```

```
        print(environment())
        print("existe o objeto:")
        print(ls())
    }
    funcao.interior(5)
    print("Dentro da funcao.exterior:")
    print(environment())
    print("existem os objetos:")
    print(ls())
}
funcao.exterior() # Lista os ambientes e objetos dentro de cada ambiente
[1] "Dentro da funcao.interior:"
<environment: 0x1080594b0>
[1] "existe o objeto:"
[1] "x.interior"
[1] "Dentro da funcao.exterior:"
<environment: 0x1080592f0>
[1] "existem os objetos:"
[1] "funcao.interior" "x.exterior"
```

11 Classes

R possui um mecanismo de classes e métodos que aplicam as funções de acordo com as classes herdadas dos objetos, no estilo de programação orientada por objetos. Existem três sistemas de classes, *S3*, *S4* e *Reference Class*. A classe *S3* é a mais simples, sem qualquer definição formal. Os objetos dessa classe pode ser criado simplesmente adicionando um atributo de classe usando a função **class**. Uma vez atribuidas os objetos passam a herdar a classe (checados pela função **inherits**), sendo que os objetos podem herdar mais de uma classe. Na classe S3 os métodos pertencem à funções genéricas junto com outras funções genéricas predefinidas (por exemplo *print*, *summary* e *plot*) que aplicam os métodos de acordo com as classes herdadas dos objetos. Quando uma função genérica é aplicada a um objeto de uma classe (por exemplo *"minhaclasse"*), o sistema procura por uma função compatível com a classe (por exemplo *"funcao.minhaclasse"*), se a encontrar aplica o método ao objeto. Se tal função não for encontrada a classe implícita ou método padrão serão tentados. É possível derivar novas classes das classes existentes e adicionar novos métodos ou criar funções genéricas próprias.

```
# Classe S3
# Função construtora da classe "minhaclasse"
minhaclasse <- function(sp, atributo){
  if(atributo<0)  stop("atributo deve ser maior ou igual a zero")
  res <- list(sp = sp, atributo = atributo)
  class(res) <- "minhaclasse" # pode ser usado a função attr()
  return(res)
}


sp1 <- minhaclasse("Sp.1", atributo = 4)
sp1
$sp
[1] "Sp.1"

$atributo
[1] 4
```

```
attr(,"class")
[1] "minhaclasse"
sp1$atributo <- -9 # A checagem só funciona quando usado função construtora


# A função print() é uma função genérica usada para mostrar os objetos na tela
# A função chama métodos particulares que dependem da classe do primeiro argumento
# É importante que o nome da função genérica seja separado da classe
# pelo símbolo de ponto (".").

print.minhaclasse <- function(obj) {
  cat("A espécie é", obj$sp, "com atributo", obj$atributo) # método próprio para print
}

sp1 # mesmo que print(sp1), chama a função print.minhaclasse()
A espécie é Sp.1 com atributo -9
```

12 Exceções

O R oferece funções para lidar com condições incomuns, incluindo erros e avisos. As funções **message** e **warning** são usadas para mostrar mensagens de diagnóstico e advertência respectivamente. Erros que interompem a execução do código podem ser atribuidos pela função **stop**. A função **try** pode ser usada como uma funçõe envelope (wrapper function) para executar uma função que pode falhar, permitindo a recuperação de erros. A função retorna o resultado normal da função caso executada sem erro, mas um objeto da classe *"try-error"* contendo a mensagem de erro e a condição de erro é retornado caso houver alguma falha. Outras funções sobre condições incomuns são encontradas em *?conditions*.

```
funcao <- function(x){
    if(x==0){
        message("mensagens de diagnóstico.")
    }
    if(x==1){
        warning("uma mensagem de advertência.")
    }
    if(x>10){
        stop("uma mensagem de erro.") # mensagem de erro fatal
    }
    return(x)
}


funcao(0)
mensagens de diagnóstico.
[1] 0
funcao(1)
Warning in funcao(1): uma mensagem de advertência.
[1] 1
funcao(11)
Error in funcao(11): uma mensagem de erro.

try(funcao(2))
[1] 2
```

```
res <- try(funcao(11))
res
[1] "Error in funcao(11) : uma mensagem de erro.\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in funcao(11): uma mensagem de erro.>
class(res)
[1] "try-error"
```

13 Paravras reservadas

Além das palavras usadas no controle de fluxo, as principais palavras reservadas no R estão listados a seguir. Lista completa disponível em *?reserved*.

```
NA # Indeterminado (Not Avaiable)
NaN # Indeterminado (Not a Number)
Inf # Infinito
TRUE # Variável lógica para verdadeiro, ou abreviação T
FALSE # Variável lógica para falso, ou abreviação F
NULL # Usado para especificar algo nulo ou vazio
... # Corresponde a zero, um ou mais argumentos reais (portanto, objetos).
```

14 Importando

Bibliotecas externas (chamados de pacotes ou *packages*) podem ser instaladas (função **install.packages**) e carregadas com a função **require** ou **library**. É possível usar funções individuais usando **::** para chamar funções dos *namespaces* dos pacotes instalados.

```
require(MASS)
Loading required package: MASS
permute::shuffle(4)
```

15 Importando e exportando arquivos

O R possui uma grande variedade de pacotes embutidos que permitem importar e exportar arquivos nos mais diversos formatos. Funções como **read.table**, **read.csv**, **write.table** e **write.csv** do pacote *utils* permitem importar (prefixo *read.*) e exportar (prefixo *write.*) tabelas de dados. As funções do pacote *grDevices* como **pdf**, **png**, **tiff**, **jpeg** permitem abrir dispositivos para exportar gráficos (a função **dev.off** precisa ser usada para fechar os dispositivos gráficos). Existem também funções para criar, abrir e fechar conexões com arquivos e URLs, lista completa disponível em *?connections*. O R sempre fica associado a uma diretório do computador, esse será o diretório de trabalho (*working directory*). Os arquivos que estão neste diretório podem ser carregados apenas pelo nome e qualquer arquivo exportado será salvo neste diretório caso não especificado o contrário. Os sistemas de caminhos (*path*), absolutos ou relativos, podem ser usados especificar a localização dos arquivos (por exemplo **./subdiretorio** e **..**) e a funções **file.choose** permitem escolher arquivos interativamente. As funções **save.image** e **load** permitem respectivamente salvar e carregar área de trabalho (*workspace*) e a função **source** importar arquivos de códigos em R.

16 Geração de números aletórios e amostragem

O R possui um conjunto grande de funções ligadas às distribuições de probabilidade, simulação de distribuições e reamostragem de dados. Amostras aleatórias e permutações podem ser realizadas pela função **sample**. As funções de densidade/massa, distribuição cumulativa e quantílica e geração de variáveis aleatórias pode realizadas para as principais distribuições de probabilidade. A lista completa disponível em *?Distributions*.

17 Guia de ajuda rápida

```
# - Adicionar comentário
? - Obter ajuda de função
?? - Relizar buscar
<- ou = - Atribuir objeto (direita para esquerda)
-> - Atribuir objeto (esquerda para direita)
<<- - Super atribuir objeto (direita para esquerda)
->> - Super atribuir objeto (esquerda para direita)
+ - Somar
- - Subtrair
* - Multiplicar
/ - Dividir
^ - Potêncializar (direita para esquerda)
%/% - Parte inteira da divisão
%% - Resto da divisão
%*% - Multiplicar matrizes
%o% - Produto diádico
%x% - Produto de Kronecker
%in% - Operador de correspondência
< - Comparar, menor
> - Comparar, maior
<= - Comparar, menor ou igual
>= - Comparar, maior ou igual
== - Comparar, exatamente igual
!= - Comparar, diferente
! - Lógico, NÃO. Inverter resultado de teste lógico
& - Lógico, critério aditivo E. Operação elementar
| - Lógico, critério aditivo OU. Operação elementar
&& - Lógico, E
|| - Lógico, OU
~ - Fórmula estatística
FALSE ou F - Argumento lógico falso
TRUE ou T - Argumento lógico verdadeiro
NA - Indeterminado (Not Avaiable)
NaN - Indeterminado (Not a Number)
Inf - Infinito
NULL - Objeto nulo
c() - Concatenar valores em vetor
factor() - Criar fator
ordered() - Criar fator ordenado
data.frame() - Criar tabela de dados (data.frames)
matrix() - Criar matriz
list() - Criar lista
rbind() - Combinar vetores por linhas
cbind() - Combinar vetores por colunas
```

```
paste() - Concatenar caracteres em sequências regulares
class() - Conferir/Atribuir classe do objeto
str() - Conferir estrutura do objeto
: - Gerar sequência numérica contínua
$ - Indexar vetores/listas pelo nome das variáveis/listas
@ - Indexar na classe S4
[] - Indexar vetores/listas
[ , , drop = FALSE] - Indexar data.frames/matrizes. Primeiro valor linha, segundo coluna
[[]] - Indexar listas
:: ou ::: - Acessar variável em um namespace
names() - Conferir/Atribuir nomes a vetores/listas
colnames() - Conferir/Atribuir nomes as linhas de data.frames/matrizes
rownames() - Conferir/Atribuir nomes as colunas de data.frames/matrizes
length() - Conferir comprimento de vetores/listas
nrow() e ncol() - Conferir dimensões de data.frames/matrizes
t() - Transpor data.frames/matrizes
seq() - Obter sequência regular
rep() - Repetir valores
ifelse() - Aplicar teste condicional
read.csv() ou read.table() - Importar tabelas
write.csv() ou write.table() - Exportar tabelas
ls() - Listar objetos da área de trabalho
rm() - Remover objetos da área de trabalho
save.image() - Salvar área de trabalho
load() - Carregar área de trabalho
if(condição) expressão - Controle de fluxo, SE
if(condição) expressão.da.condição  else  expressão.alternativa - Controle de fluxo, SE/SENÃO
for(variável in sequência) expressão - Controle de fluxo, LAÇO/LOOP.
while(condição) expressão - Controle de fluxo, ENQUANTO
repeat expressão - Controle de fluxo, REPETIR
break - Controle de fluxo, PARAR
next - Controle de fluxo, PRÓXIMO
function(lista.de.argumentos) expressão - Funções. Criar função
return(valor) - Funções. Resultado/returno da função
message() - Funções. Criar mensagens de diagnóstico
warning() - Funções. Criar mensagens de advertência
stop() - Funções. Criar mensagens de erro
try() - Funções. Tentar função envelope
```

18 Conclusão

O objetivo deste texto foi introduzir alguns conceitos básicos da linguagem R. Os principais aspectos da sintaxe, operadores, tipos de dados, estruturas dos objetos, controle de fluxo e desenvolvimento de funções foram mostradas e exemplificadas. Espero que este texto tenha sido útil e, por favor, avise-me se tiver dúvidas ou sugestões sobre este texto.

19 Mais informações

Outros textos e tutoriais sobre R podem ser encontrados em https://vanderleidebastiani.github.io/tutoriais.

20 Referências

- Crawley, Michael J. 2007. **The R book**. John Wiley & Sons, Chichester.
- DataMentor. 2018. **Learn R Programming**. https://www.datamentor.io/r-programming/
- R Core Team. 2018. **R Language Definition**. https://cran.r-project.org/doc/manuals/R-lang.html